Разработка приложений на платформе .NET

Лекция 14

Основы Windows Presentation Foundation
XAML

Литература

- Мэтью Мак-Дональд. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов (2013)
- Натан Адам. WPF 4 Подробное руководство (2011)

Обработка графики

- Графика в **Windows Forms**
  - Рисование силами ОС (использование Windows API)
  - User32 – внешний вид окон, элементов управления (кнопок, текстовые поля и т.д.)
  - GDI / GDI+ - рисование фигур, текста, изображений.
  - Обработка графики Центральным процессором (CPU)
- Графика в **WPF**
  - Использование DirectX. Отображением элементов занимается DirectX.
  - Ускорение за счет аппаратных средств графической подсистемы (GPU).
  - Совсем не используется системный модуль GDI / GDI+.
  - User32 – по прежнему используется, но в минимальных количествах. Обработка и маршрутизация ввода, определение участка экрана, принадлежащего приложению.
- Следствия использования DirectX в **WPF**
  - Более богатые графические возможности
  - Поддержка 3D графики
  - Поддержка произвольной анимации и мультимедиа
  - При проектирование интерфейса WPF обычно используется векторная графика

Единицы измерения

- В Windows Forms
  - пиксель
- В WPF
  - независимая от разрешения единица измерения (равная 1/96 дюйма)
  - Элементы выгладят одинаково на экранах с разным разрешением. На экранах с более высоким DPI (точек на дюйм экрана) интерфейс более четко прорисовывается, а не уменьшается в размерах
  - Используется системная установка DPI
  - Размер элемента пересчитывается на текущее значение DPI в системе.
  - При дробном значении пикселя используется сглаживание.

Возможность WPF

- Пример:
- В **Windows Forms** разные кнопки
  - Кнопка с текстом – просто
  - Кнопка с текстом и рисунком – сложнее
  - Кнопка с видео – неподъемно...
- В **WPF** и **XAML** все 3 кнопки делаются одинаково просто. Это возможно благодаря возможности вкладывать одни элементы управления в другие
  - Это применимо не только к кнопкам, но и почти ко всем другим элементам управления

Демонстрации

Пример WPF-приложения
Анимированная кнопка с видео

Уровни аппаратного ускорения WPF

- Уровни аппаратного ускорения:
  - 0 – Видеокарта не представляет никакого аппаратного ускорения (DirectX 9 и ниже)
  - 1 – Видеокарта поддерживает DirectX 9, но недостаточно ресурсов
  - 2 – Видеокарта поддерживает DirectX 9 и достаточно ресурсов
- Об уровне можно узнать по значению переменной RenderCapability.Tier
- (RenderCapability.Tier >> 16)
- Определяется при первом при первом старте инфраструктуры WPF
- Если видеокарта не поддерживает аппаратного ускорения, то недостающие возможности обеспечиваются за счет CPU

Особенности WPF

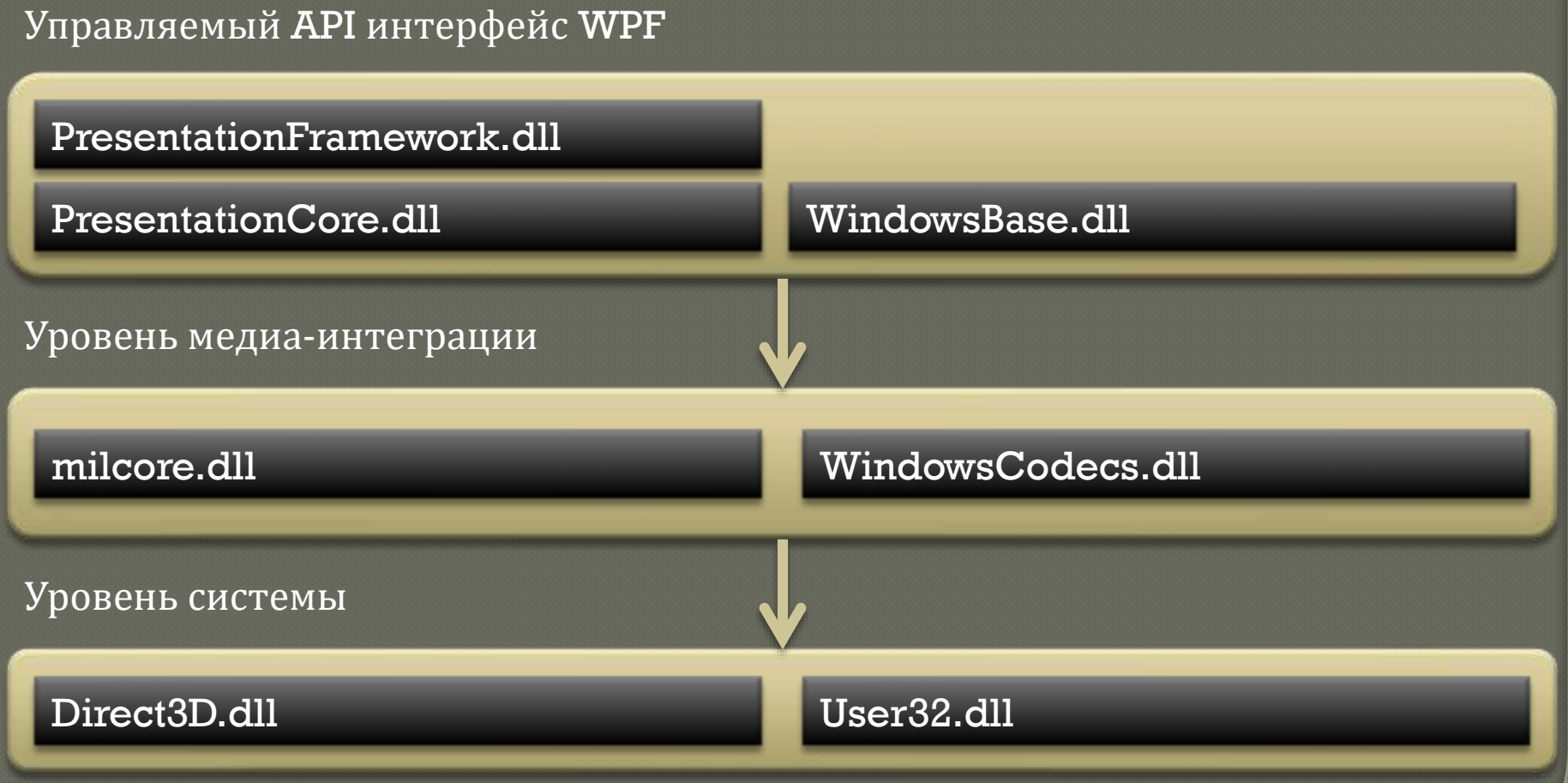
- Аппаратное ускорение
- Независимость от разрешения
- Отсутствие фиксированного внешнего вида
- Декларативное описание пользовательского интерфейса (**XAML**)
  - **XAML -** декларативный язык описания интерфейса
- Рисование на основе объектов
- Поддержка аудио и видео
- Продвинутые возможности отображения текстовых документов

- Анимация. Декларативное описание анимации
- Система команд
- Поддержка стилей, тем и шаблонов
- Привязки (**Binding**)

Идеология WPF

- Разделение логики и оформления
  - Бизнес логика – **C#**
  - Оформление (интерфейс) – **XAML**
- Разделение задач
  - Логика – программист
  - Интерфейс – дизайнер (используя, например, **Expression Blend**)
- Подходы для создания пользовательского интерфейса
  - Декларативный (**XAML**)
  - Императивный подходы (**C#**)
- Независимость от разрешения экрана
  - Произвольное изменение размеров окон
  - Автоматическая адаптация под содержимое (например, локализованные ресурсы)
  - Гибкая компоновка пользовательского интерфейса
- Отложенная модель отрисовки
  - содержание – разработчик
  - отрисовка – система

Архитектура WPF



XAML

eXtended Application Markup Language

расширенный язык разметки приложений

Что такое XAML

- **XAML – eXtended Application Markup Language** – расширенный язык разметки приложений
- Основан на **XML**. Расширяет его
- Это декларативный язык, описывающий структуру графического интерфейса, стили и сценарии
- Декларативность – описание структуры и свойств, без кода

Особенности WPF

- Использование **XAML** для
  - Определения структуры **UI** (типа **HTML**)
  - Задания стилей (типа **CSS**)
  - Анимации и мультимедиа
  - Трехмерной графики и анимации
- Использование **C#** для
  - Обработки событий
  - Логики приложения

Разновидности XAML

- Применение **XAML**
  - **Windows Presentation Foundation (WPF)**
  - **Silverlight**
  - **Workflow Foundation (WF)**
  - **Windows Communication Foundation (WCF)**

  - **XAML** может использоваться в любой другой предметной области для декларативного описания, используя пользовательское множество объектов

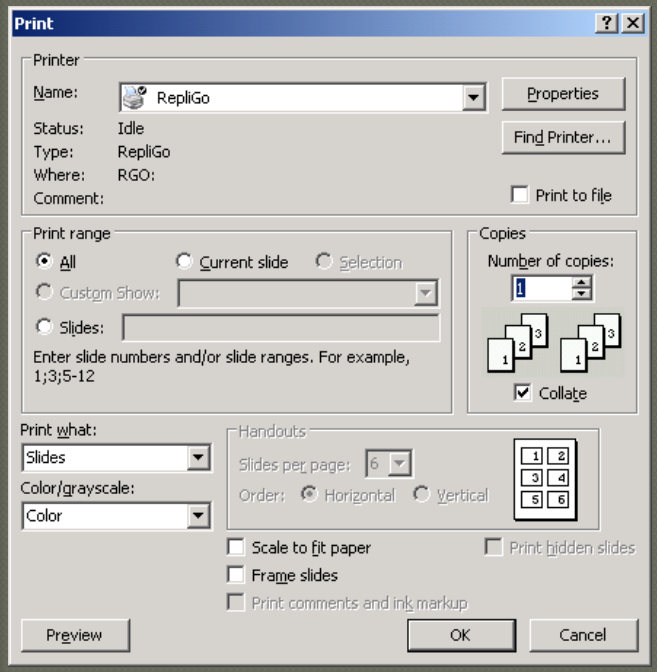
- **WPF** использует **XAML**, но может и обойтись без него
- **XAML** используется и в других областях, не только в **WPF**

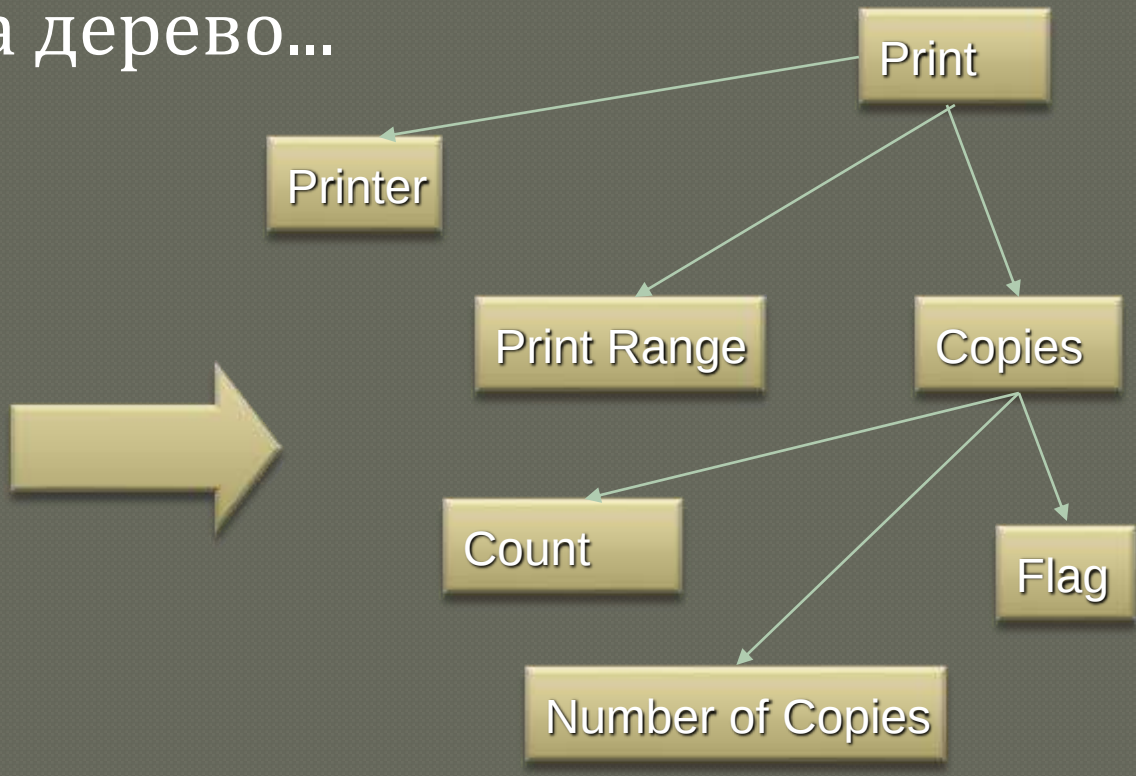
Демонстрации

Hello, WPF World!

Структура интерфейса

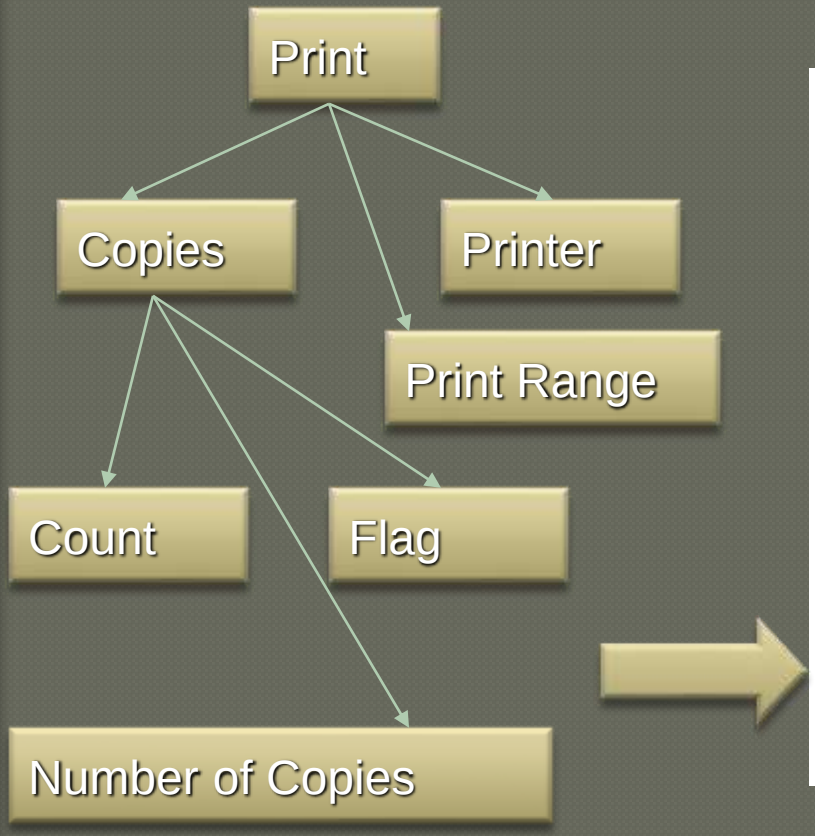
очень похожа на дерево...





Для описания деревьев

- Используют **XML – eXtended Markup Language**




```
<Window Title="Print">

 <GroupBox Title="Copies">
  <CheckBox Title="Collate"
            IsChecked="true"/>
  <Label Title="Number of Copies">
  <Spinner MinValue=1 MaxValue=1000 />
 </GroupBox>

 <GroupBox Title="Print">
 </GroupBox>
</Window>
```

Элементы UI

- Элементам UI соответствуют
  - С одной стороны – элементы XML
  - С другой стороны – классы .NET
- Имеется соответствие между .NET и XAML

Соответствие XAML←→ C#

XAML	**C#**
Элемент	Объект класса
Атрибут	Свойство
Вложенность	Спец. свойство
-	Метод
Триггер	-
Сеттер	-
Атрибут-событие	Подписка на событие

Как это работает

- Компиляция
  - XAML компилируется в BAML и добавляется в качестве ресурса к сборке
  - Создается частичный класс реализующий метод InitializeComponent(), который:
    - загружает BAML
    - создает объекты (по структуре BAML)
    - привязывает объекты к переменным (каждому именованному элементу в XAML в частичном классе создается соответствующее поле)
    - привязывает обработчики события
  - Частичные классы объединяются
- Исполнение
  - Исполняется C#-код
- Удобство
  - Благодаря декларативному подходу создавать интерфейс на XAML значительно удобнее, чем на C#

Пространства имен XAML

- xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" – основное пространство имен **WPF**. Охватывает все классы **WPF**, включая все классы элементов управления. По умолчанию задается как основное пространство имен в **XAML**.

- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" – пространство имен **XAML**. Содержит служебные свойства **XAML**, которые влияют на то как интерпретируется документ.

- Можно добавлять и другие пространства имен для использования в **XAML**.
- Синтаксис:
  - xmlns:*Префикс*="clr-namespace:*ПространствоИмен*;assembly=*ИмяСборки*"
    - *Префикс* – префикс **XML**, который будет использован для указания пространства имен в разметке **XAML**
    - *ПространствоИмен* – полное название пространства имен **C#**
    - *ИмяСборки*– имя сборки, в которой описано пространство имен (без **dll**).
  - xmlns:Sys="clr-namespace:System;assembly=mscorlib"
  - xmlns:Col="clr-namespace:System.Collections.Generic;assembly=mscorlib"

Свойства

- **Xml** атрибут трактуется как свойство объекта
- **<TexBlock Text=“Привет”/>** - все понятно: **Text** – типа **string**
- **<TexBlock Margin=“2,3,3,1”/>** - Свойство **Margin** имеет тип **Thickness.** Непонятно как по строке создать тип **Thickness** и задать ему параметры.
- Для этого анализатор **XAML** использует конвертеры типов. Класс – конвертер указывается с помощью атрибута **TypeConverter** для свойства (например, **Margin**) или для класса (например, **Thickness**). Анализатор **XAML** ищет конвертер, преобразует с его помощью строку в нужные тип и присваивает результат свойству.

Сложные свойства

- Трудности при задании в виде строки (например, коллекции)
- Задаются с помощью синтаксиса элемент-свойство
  - <РодительскийЭлемент.ИмяСвойства> ….
  - </РодительскийЭлемент.ИмяСвойства>
- По точке элемент распознается не как класс, а как задание сложного свойства

```
<Grid>
    <Grid.Background>
      <LinearGradientBrush>
        <GradientStopCollection>
          <GradientStop Offset="0" Color="Orange"/>
          <GradientStop Offset="1" Color="Yellow"/>
        </GradientStopCollection>
      </LinearGradientBrush>
    </Grid.Background>
</Grid>
```

- Может применяться для задания любого свойства

Присоединенные свойства

- Свойства, которые определены в одном классе, а применяются во многих других классах, не связанных наследованием с определяющим классом.

- Синтаксис:
  - ОпределяемыйТип.ИмяСвойства=“значение”

```
<Grid>
    <ComboBox Grid.Row="0" …./>
</Grid>
```

Именование элементов

- Свойства **Name** и **x:Name**
- При задании имени в автоматически генеренной части класса создается поле с таким именем и типом соответствующим типу элемента. Также в методе **InitializeComponent** будет создан код для задания этого поля созданным объектом при анализе **BAML**.
- В отличии от **Windows Forms**, элемент в **WPF** не обязан иметь имя. Имя необходимо задавать, если элемент предполагается использовать в коде или необходимо ссылаться на элемент в **XAML**

Вложенные элементы

- Каждый элемент сам решает, как поступать со своими вложенными элементами
  - Если родительский элемент реализует интерфейс **IList**, анализатор **XAML** вызывает метод **IList.Add**(), передавая вложенный элемент.
  - Если родительский элемент реализует интерфейс **IDictionary**, анализатор **XAML** вызывает метод **IDictionary.Add**(), передавая вложенный элемент. При этом необходимо задать свойство **x:Key** для каждого вложенного элемента.
  - Если родительский элемент помечен атрибутом **ContentProperty**, то анализатор использует дочерний элемент, чтобы установить указанное свойство

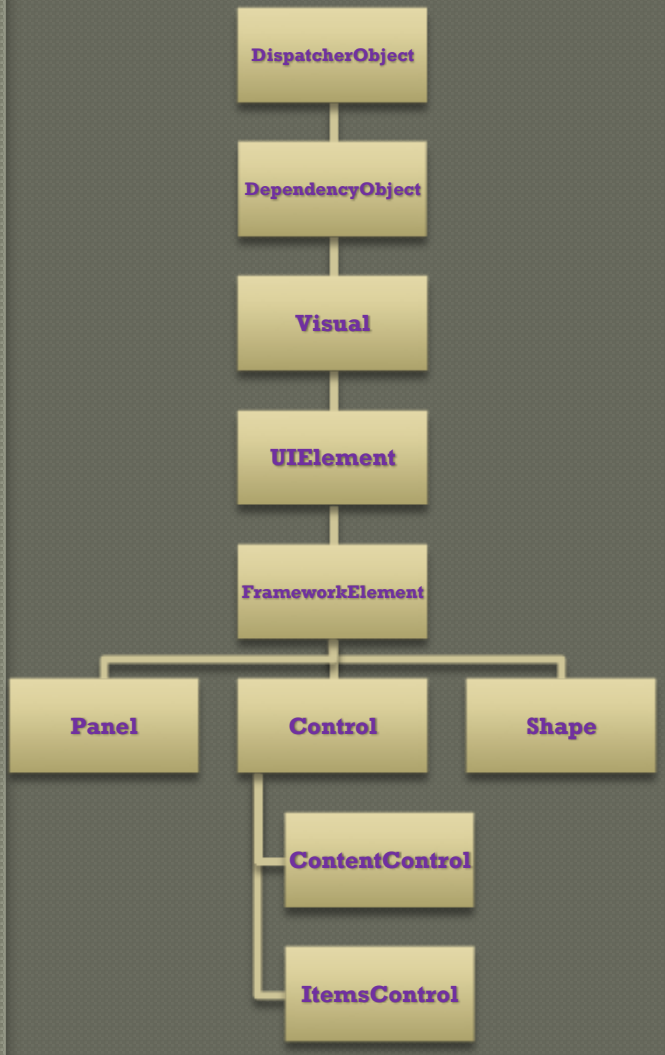
```
<Button>Текст кнопки</Button>
<ListBox>
    <CheckBox Content="Red"/>
    <CheckBox>Green</CheckBox>
    <CheckBox>Blue</CheckBox>
</ListBox>
```

Специальные символы

- Определены два метода записи
  - &имя_сущности;
  - &#UTF_код_символа;
- Сущности:

Символ	Пример	Пример
&	&	&
<	<	>
>	>	<
‘	'	'
“	"	"

Иерархия классов



– **STA.** Управление потоками

– Свойства зависимости

– Отображение

– Компоновка, события, фокус, ввод

– Варианты компоновки, привязки, стили, анимация

Panel – базовый для всех контейнеров компоновки
Control – базовый для всех контролов. Поддержка шаблонов
Shape – базовый для всех графических фигур

ContentControl – Отображение одного содержимого
ItemsControl – Отображение коллекции

Основные классы

- **System.Windows** – пространство имен
- **Application**
  - Приложение в целом
  - События приложения
  - Цикл обработки сообщений
- **Window**
  - Окно верхнего уровня приложения
  - События окна
  - Основа логики **WPF**

Application

- Свойства
  - **StartupUri** – **XAML**-файл приложения
  - **MainWindow** – главное окно
  - Указан **StartupUri** – окно назначается автоматически
- Методы
  - **Run()** – запуск цикла обработки сообщений
- События
  - **Startup** – запуск приложений
  - **Exit** – выход из приложения
- Навигационные приложения

Window : ContentControl

- Свойства
  - **Content** – наполнение окна
  - ...
- Методы
  - **Show()** – показать окно
  - **ShowDialog()** – показать окно в модальном режиме
  - **Close()** – закрыть окно
  - **Hide()** – спрятать окно
- События
  - **Loaded** – окно загружено
  - **Closed** – окно закрыто
  - **Closing** – окно закрывается